Second Source Component Probing on Device Tree Platforms

EOSS Seattle 2024

Chen-Yu Tsai, ChromeOS

Disclaimer

- Examples mainly focused on ChromeOS devices
- Some experiments done
- Some code published
- Open problem

Agenda

- Background
- Current status
- In-kernel component probing
- In-kernel device tree modification
- Per-variant overlays

Background

Component Second Sourcing

- Various hardware components are selected at the design stage
- Multiple sources for a given component could be selected
- Components could be swapped out later on due to shortages or phase-outs
- Second source components are only functionally similar
  - Provide the same function
  - Unlikely to be pin compatible
    - Unless modularized through a common connector
      - Common for display panels, touchscreens and touchpads
  - Likely requires different power sequencing and drivers
  - All changes expected to be solved through software

Device Trees

- A minimal description of the (non-discoverable) hardware
  - Enough information for the OS to enable and use the device
  - Excludes enumerable devices such as USB, PCIe, and SD/MMC/SDIO …
  - Unless non-discoverable properties such as regulator supplies or GPIOs for them exist
- Device trees selected with the root compatible string (just on ChromeOS?)
  - One device tree could be used for multiple devices
  - More component combinations lead to more compatible strings
  - `.../mediatek/mt8186-corsola-magneton-sku393216.dts`:

```
compatible = "google,steelix-sku393219", "google,steelix-sku393216",
             "google,steelix", "mediatek,mt8186";
```

ChromeOS Bootloader

- ChromeOS uses FIT images for delivering the kernel
- FIT image includes
  - kernel image
  - a bunch of device trees and overlays if any
  - a bunch of configurations
    - Combinations of kernel image + one base DTB + zero or more overlays
- Configuration compatible derived from board level compatible string in each base DTB
- Device trees are always bundled with the kernel, not the firmware

ChromeOS Bootloader

- Bootloader knows its device family "Kakadu" at build time
- Bootloader reads ADC value to figure out SKU and revision numbers
  - Or some other data source
- Bootloader generates a list of compatible strings to try
  - "google,kakadu-rev2-sku22"
  - "google,kakadu-rev2"
  - "google,kakadu-sku22"
  - "google,kakadu"
- Bootloader goes through the above list one by one
  - Finds a configuration in the FIT image with a matching compatible string

Related Work

- Shipping Multiple Devicetrees: How to Identify Which DTB Is for My Board? - Elliot Berman, Qualcomm
  - Same problem space, different constraints
    - ChromeOS uses FIT images; Qualcomm has something different
      - FIT images provide one extra layer of mapping
  - [PATCH v2 0/2] Add board-id support for multiple DT selection

Goals

- Reduce overall kernel FIT image size
- Reduce number of device trees to maintain
- Try to move away from SKU + revision compatible strings
  - At least in the device tree files

The Current Mess

Non-conflicting Probable Components

- Manufacturer switches to a new I2C trackpad or touchscreen solution
- New component's I2C address does not conflict with existing usage
- No new hardware SKU identifier allocated
- Add a new device node and let the kernel drivers sort out which one is actually there
- Modules can be easily swapped around (for development or repair)

Conflicting Component Change

- Manufacturer switches to a new I2C trackpad or touchscreen solution
- New component's I2C address conflicts with existing usage
- New hardware SKU identifier allocated
- Duplicate a new device tree with new compatible string and component node replaced

Hard-to-discover Component Change

- Manufacturer switches to a new MIPI display panel
  - MIPI display panels have wildly varying power sequences and commands
- No separate addresses to differentiate components
- New hardware SKU identifier allocated
  - Or maybe the component contains a strapping that maps into the identifier
- Duplicate a new device tree with new compatible string and component node replaced

Enumerable Component Change

- Manufacturer has two USB cameras with different resolutions
- Needs hardware identifier to check hardware variant against camera resolution
- New hardware SKU identifier allocated
- Duplicate device tree with new compatible string and **nothing else changed**
- Does not need any special handling
- Contributes to the increase in number of device trees
- Avoided by moving to a separate identifier
  - Nothing can be done for devices already on the market

The Current Mess

- Multiple workarounds for kernel device driver probing in the device tree
- Mass duplication of device trees with only minor variations

I2C Component Probing

I2C Bus and Components

- Not discoverable
  - No bus enumeration
- Possibly probable
  - Given a list of possible addresses, software can probe which of them respond
  - Components likely do not have a "chip ID" register that can be used to verify the probe result
    - DT says address 0x10 is a touchscreen, but actual hardware is a sensor
    - I2C HID devices provide a fixed format descriptor
      - But the address varies

Current Status

- DT has a number of possible components on the I2C bus
- Kernel binds each device to respective driver and probes
  - **Concurrently!**
  - Driver made to probe for component existence before IRQ request
  - Reset GPIO line described with pinctrl properties tied to I2C bus
    - Instead of `reset-gpios = <...>` in the component device node
  - Regulator supplies always on
    - Possibly by design
    - Concurrent on/off requests likely don't work either

Probing in Firmware

- Doable
  - Done in Thinkpad X13S w/ Qualcomm SoC for the trackpad
    - Doesn't pass result to OS :-(
- Firmware needs some way to pass result to OS
  - Modifying the DT could couple the firmware too tightly
    - Bindings might not have been fully vetted when the firmware is locked down
    - Devices might be under embargo and can't be upstreamed yet
    - Device tree paths or compatible strings may differ downstream
  - Requires firmware be updatable for existing devices
- But: firmware should do just enough to boot the OS
  - Firmware runs at minimal frequency and only on a single core
  - Firmware is hard to update, if updated or updatable at all

In-kernel I2C Prober

- Coordinated probing by a platform-specific driver
  - Tries to do power sequencing just once
  - Avoids resource conflicts
  - Components disabled (wait for probe) in device tree
  - Simple I2C read transfer to probe for component existence
  - Only components that are successfully probed are enabled and used by the kernel
- RFC series v3
  - https://lore.kernel.org/linux-arm-kernel/20231128084236.157152-1-wenst@chromium.org/

In-kernel Device Tree Modification

Device Variants

- Bootloader inserts identifier into device tree
  - Bitfields or range of values for each component change
- Device tree lists all possible components
- Kernel looks at bits in identifier, and enables corresponding component(s)
  - Or read the strappings directly
- RFC series v2
  - https://lore.kernel.org/linux-arm-kernel/20231109100606.1245545-1-wenst@chromium.org/

- DO NOT GENERATE DEVICE NODES ON THE FLY

Trade Offs

+ Reduces number of DTs
- Pushes some logic into the kernel
- DT validation errors due to incomplete OF graph
    - For display panels and non-USB camera sensors

Per-Variant Overlays

ChromeOS Hardware Designs

- One (or more) reference designs / platform
  - Maybe one or more design deviations
    - One or more projects per design
      - One or more SKUs per project

ChromeOS Hardware Designs

- MT8186 SoC platform
  - Kingler reference design (no actual projects)
  - Krabby reference design
    - `mt8186-corsola-krabby.dtsi`
    - Tentacruel project
      - `mt8186-corsola-tentacruel-sku*.dts`
  - Steelix design (done by manufacturer)
    - `mt8186-steelix.dtsi`
    - Steelix project
      - `mt8186-corsola-steelix-sku*.dts`
    - Magneton project
      - `mt8186-corsola-magneton-sku*.dts`

Currently: One DTB Per SKU

- 10 DTBs for MT8183 Kukui family
- 17 DTBs for MT8183 Jacuzzi family
- 10 DTBs for MT8186 Corsola family
  - More to come
- Lots of duplication from .dtsi files within each family

Overlays for Minor Differences

- Base DTBs for each device
- Overlays for per-SKU variations
- Convert
  - `mt8183-kukui-kodama.dtsi`
  - `mt8183-kukui-kodama-sku{16,32,272,288}.dts`
- To
  - `mt8183-kukui-kodama.dts`
  - `mt8183-kukui-kodama-sku{16,32,272,288}.dtso`
  - `mt8183-kukui-kodama-sku16.dtb =`
    `mt8183-kukui-kodama.dtb + mt8183-kukui-kodama-sku16.dtbo`

Overlays for Minor Differences

- Significant size increase for individual DTB files
  - Symbol and fixup tables
    - Could be dropped after overlays are applied?
  - `/omit-if-no-ref/` ignored
  - Extra phandle properties created
+ Significant size savings for bundled images, i.e. FIT images
  - Bundle individual components
    - Base DTB
    - DTB overlays
  - Bootloader applies overlays to produce final DTB
  - 6.61% MT8183 DTB bundle size increase if final composite DTBs used
    - Only Kodama devices converted
  - 6.7% MT8183 DTB bundle size decrease if individual components bundled and de-duped

Overlays for Devices Within a Family

- Base DTBs for each family
- Overlays for per-device and per-SKU variations
- Convert
  - `mt8183-kukui.dtsi`
  - `mt8183-kukui-kodama.dtsi`
  - `mt8183-kukui-kodama-sku{16,32,272,288}.dts`
- To
  - `mt8183-kukui.dts`
  - `mt8183-kukui-kodama.dtso`
  - `mt8183-kukui-kodama-sku{16,32,272,288}.dtso`
  - `mt8183-kukui-kodama-sku16.dtb = mt8183-kukui.dtb + mt8183-kukui-kodama.dtbo + mt8183-kukui-kodama-sku16.dtbo`

Issues

- Changes how DTS files are managed
- Doesn't reduce the number of files
- No proper compatible string for the base DTB(s)
- DTB size increase
- /delete-property/ and /delete-node/ won't work
  - Two in-tree .dtso files have /delete-property/
- DT validation might not work as intended
  - Composite DTB is not validated
- Needs bootloader support to get deduplication benefits

Per-Component Overlays

Variants vs Components

- Variant = audio codec X + trackpad Y + touchscreen Z + … etc. components
- 2 of X × 3 of Y × 5 of Z = 30 device trees
- One overlay per variant is still a lot of tiny files
- What if the bootloader knew about the individual components?

Variants vs Components (cont.)

- Variant = audio codec X + trackpad Y + touchscreen Z + … etc. components
- Bootloader can identify feature set and knows to apply specific overlay for each component
  - audio-codec-X.dtbo
  - trackpad-Y.dtbo
  - touchscreen-Z.dtbo
  - On top of base.dtb
- No more variant overlays with just "compatibe = …" and "model = …"
- 2 of X + 3 of Y + 5 of Z = 10 files for components
  - Reduces number of source files

Feature Overlays (simplified)

- Kernel produces base DTB + feature overlays
- Bootloader still uses "`<family>-<rev>-<sku>`" compatible string to pick configuration to use
- FIT image packer maps compatible strings to separate DTB + overlay combinations
  - SKU 0 = base + audio codec X ; SKU 1 = base + audio codec Y
  - Where do we keep the mappings?
  - Reference overlays by file name?
    - Or embed the metadata in the overlay maybe?

Feature Overlays

- FIT support for extension boards / overlays proposal, Simon Glass
  - Apply overlays included in FIT image based on firmware generated compatible strings
- Open problems not covered by the proposal
  - Needs a repository for metadata (mapping and dependencies)
    - Can this be internalized into the device trees?
    - Firmware needs to map feature to extension compatible strings
  - Same board compatible strings for many combinations
    - Same issue seen with current in-tree overlays under freescale/
    - Do we still build composite DTBs?

Summary

Summary

- Coordinated probing for some component classes
- Design and component changes result in many device variants and separate DTB files
- Various approaches to reducing duplication
  - All come with trade-offs
- How should we proceed?

Thank You